A TOOL TO AUTOMATE SOFTWARE PROJECT ESTIMATION FROM A PROJECT MANAGEMENT PERSPECTIVE

By

GOPAL N. KULKARNI

Bachelor of Engineering

Karnatak University

Dharwad, India

1984

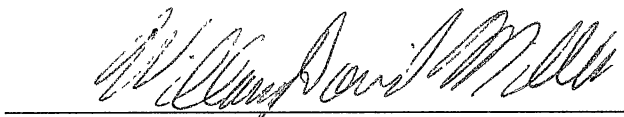
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the degree of
MASTER OF SCIENCE
December, 1991

A TOOL TO AUTOMATE SOFTWARE PROJECT

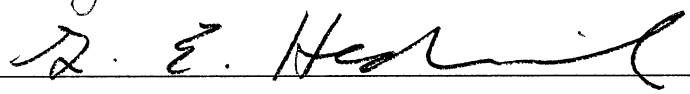
ESTIMATION FROM A PROJECT

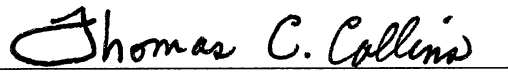
MANAGEMENT PERSPECTIVE

Thesis Approved:



Thesis Adviser





Dean of the Graduate College

ACKNOWLEDGEMENTS

I profoundly thank my graduate advisor Dr. David Miller for his unstinted help and guidance. His constructive criticism helped me in gaining confidence during my graduate program. My sincere thanks to Drs. G. E. Hedrick and J. P. Chandler for serving on my graduate committee. Their suggestions and support were very helpful throughout the study. I thank Dr. M.Samadzadeh, for the help given to me in my thesis.

I would like to express my gratitude to Learmonth & Burchett Management Systems, Inc., for sponsoring this project in part. In particular, I offer my thanks to Mr. John Bantleman, Mr. Rick Plezcko and Mr. David Hsieh for their help. My special thanks are extended to Mr.Sridhar Chandrashekar and Mr. Manohar Rao for helping me in everything, from designing to debugging. I thank my wife Mrs. Shubhadaini Joshi for being so understanding and cooperative.

I will be failing in my duty if I did not express my gratitude to my father Sri. Narayanrao, my mother Sow. Laxmibai, and my brother Raghvendra without whose support, and sacrifice my higher education would not be possible.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

NOMENCLATURE

CASE. Computer-Aided Software Engineering.

CAPE. Computer-Aided Project Engineering.

Direct Manipulation. The use of a pointing device to perform actions on objects.

Double-Click. To press and release a mouse button within a user-defined time limit without moving the mouse pointer off the choice [IBM SAA/CUA Guide, 1990].

Entity. Is a component of the system that can be identified distinctly. Entities are classified into entity types. Each type represents a group of components which share similar characteristics and perform similar functions in the system [OPRR API 1991].

Estimating Formula. To estimate resources at any level in the WBS, a formula is used. This formula mixes and matches various estimating factors involved for that activity.

Estimating Factor. Is a quantitative expression of considerations given in the process of estimation.

Estimating Model. It is a mathematical description of aestimating method applied to a particular project. It is expressed as a set of formulas.

Function Point. A measure of complexity of what is to be delivered in a project.

Icon. A pictorial representation of an object or a selection choice. Icons can represent objects that users want to work on, or actions that users want to perform.

Iconised. An Object stored or represented as an icon.

Menu. A component of a dialog designing consisting of a screen which can display options and receive control input.

Multiple Document Interface. An interface style that allows users to view many objects at the same time or the same object many times within one primary window [IBM SAA/CUA Guide, 90].

CHAPTER I

INTRODUCTION

An estimate is a prediction "based on a probabilistic model, not a deterministic one--that is, the quantity being estimated can take not just one value but several values, with some more likely to be more correct than others" [Burrill and Ellsworth, 1980]. Software estimation involves estimation of effort, time, cost, personnel, etc.

Software Engineering encompasses a variety of technical methods, a set of management procedures, and a suite of automated tools (often called CASE - Computer-Aided Software Engineering) that enhance our ability to build effective computer-based systems [Pressman, 1988]. A project goes through several stages referred to as the software development life cycle. The partitioning of the project into several small modules is referred to as a Work Breakdown Structure (WBS). An estimate is characterized by the following information [Donald Reifer, 1991]:

1) Statement of Work: An identification of the tasks to be performed as part of the development process (WBS).

2) End Product List: A list of the products to be

generated and their format (Deliverables) .

3) Risk Assessment: An identification of the risk factors along with their relative impact on cost and schedule (risk Analysis) .

4) Definitions: An explanation of the key terms used in the estimate (number of sourcelines of code, staff-month of effort, etc.).

5) Assumptions: A list of any primary assumptions upon which the estimate is based (productivity assumptions).

6) Schedule : An identification of any schedule constraints (task dependencies).

7) Estimated Resources: A prediction of what resources are needed to complete the job as enumerated above.

A tool can produce an estimate with the above characteristics if it is an integral part of a set of tools that perform wide ranging tasks like Life Cycle Building, Risk Analysis, Scheduling, etc. LBMS Inc., of Houston is developing such an integrated set of PC based tools known as Project Engineer. Project Engineer is designed to perform all aspects of project management like Life Cycle Building, Estimating, Resource Management etc. The Estimator module is one of the tools provided by the Project Engineer. The Estimator tool is the topic of this study and henceforth will be referred to as the Project Engineer ESTIMATOR. The Project Engineer ESTIMATOR is sponsored and funded by LBMS Inc., of Houston. The tool is being developed as a part of

this study. The term "estimate" or "estimates", is used in a generic sense for cost, time, and effort. The following paragraphs explain some problems associated with estimation methods [Function Point Analysis with LSDM, 1988].

It is often reported that many software development projects are completed late and over budget. Accepting this statement as premise, there are two conclusions which are commonly drawn from it. Some argue that it means that many software development projects are done badly; others maintain that the estimation techniques used during development projects are thereby shown to be inadequate.

Adopting the middle ground, it is clear that there is some truth in both of these possibilities. What is very much needed is a way of disentangling the two issues. If a reliable, standard way of estimating resource requirements were available, then it would be possible to determine which projects were really using more resources than necessary, and for allocating appropriate amounts of resources in the future.

Estimating is intimately connected with productivity measurement. It is only possible to estimate future requirements if one or more objective, reasonably accurate ways of measuring present productivity are available. In addition productivity measurement is also a means to other ends than providing the basis for more accurate estimates. A very high priority final objective is productivity improvement. Broadly, this may be achieved by isolating

those factors which contribute positively or negatively to productivity and promoting or eradicating them. Identifying such factors is virtually impossible without objective productivity measurement methods.

In the section on Project Estimation some popular estimating methods are delineated. Object based estimation is the most common method used in practice. The object based estimation method has been chosen as the method of estimation for the implementation in the ESTIMATOR.

Project Estimation

A project estimation provides an objective basis for projecting the amount of effort required to carry out Systems Engineering tasks. The estimating process should allow estimating model verification, and subsequent tuning, in order to compare actuals against planned values via an appropriate control model [LBMS Estimating Guidelines, 1989].

There are many estimating models and methods. Choice of models depends on the application, and the choice of method depends on the access to history files, management preferences, and on the company culture. The following is an overview of estimating methods:

1) Object Based: Estimates are developed by breaking down the job into detailed tasks and then having those who will do the work predict the time and effort involved. Costs are

then summed to obtain the total estimate at the project level and at any intermediate level. Object based estimation is based on the premise that the effort required to produce a deliverable is the sum of the effort associated with each of the deliverable's objects. An example is given in Figure 1. page 14. which is a diagram consisting of rectangular boxes joined by lines. The drawing of the diagram illustrates the task of describing a required data model which, is a deliverable. The more boxes and lines on the diagram, the greater the effort required to create and describe the diagram, as shown in Figure 2. page 15.

2) Function Point Analysis: Function points are used to measure system size as a component of productivity measurement [Zells, 1990]. The process starts at a point when a comfortable amount of analysis and design has been completed. The estimators classify and count raw function points. For each transaction cataloged for a system, for I/O fields and for entities referenced, are identified and given a number of points. Since each such item is called a function in this context, these points are called function points. They are summed for each transaction as a measure of its "size", and an estimate of the "size" of the complete system is then obtained by adding together the points contributed by all transactions. By comparing the resources taken, or being taken, to develop "size", an objective measurement of productivity can be obtained for the

development of the system. Moreover, an estimate of the resources required for the development can be obtained by performing the same analysis before commencement [Function Point Analysis with LSDM, 1988].

3) Mathematical Models: Estimates are developed using mathematical models which vary estimates as a function of several parameters (such as number of source lines of code etc.). Again, estimates can be made at any level of the work breakdown hierarchy using this approach.

4) Expert Judgment: Software engineering consultants, based on their experience, predict estimates for projects. The accuracy of the estimate depends solely on the expertise of the consultant, so estimates are developed by different people using different approaches. Consensus can then be reached as these estimates are refined for each project to arrive at final estimates.

The Object-based estimation method is the most popular method. In this study only the object-based estimation is implemented. The Function-point method is proposed to be implemented in the next version of the ESTIMATOR.

Project Estimation Tools

Several project estimation tools exist in the marketplace, but most of these tools only perform spreadsheet like calculations with some scheduling activities. They assume that estimation is done off-line. Existing tools are inflexible and do not allow tuning of estimating models. Most of the tools do not have a windows style graphical user interface (GUI). Above all, these tools do not allow different estimating model templates to be manipulated and saved for future use.

While planning is essential, it does not, by itself, produce technical deliverables. Project Management Tools (including estimating tools) do not aid the project manager in all the activities of project estimation and planning. There is a need for an estimating tool that helps the Project Manager in managing the process of estimation at all phases of a software life cycle.

Computer-Aided Project Engineering

There are two aspects to a software project: techniques and planning [Hsieh, 90]. While the techniques help in implementing the project, planning aids in designing the project to be properly conceived and executed. As mentioned in the previous section, there are several tools to automate the techniques part (in particular the CASE tools that

support the Programmer/Analyst/Designer). However, the planning and management aspects, along with the automation of software methodologies, usually are not satisfactorily supported by existing tools.

While CASE tools automate the production of technical deliverables, CAPE tools automate the production of planning deliverables. Delivery of planning deliverables usually is accomplished manually by project managers, which makes manipulation and maintenance cumbersome. A tool to perform these management aspects could increase the productivity of the project manager, while providing a proper foundation for the project from the start. Currently, support for the Project Manager is limited. The ESTIMATOR is part of an integrated set of CAPE tools.

Statement of the Problem

Many software projects overrun their estimated cost and schedule, and do not meet the expected quality standards. Few projects are completed within reasonable limits of the original estimates. Such overruns are due to poor estimation by software developers and managers. So what is the cause of inaccurate estimates, and how can the error be reduced?

To begin, competent estimators, based on their sheer experience and expertise, often get promoted to positions where their skill is no longer required. Then, newly

selected estimators rarely find adequate procedures or skilled consultants from whom they can learn. This is compounded by the fact that there is rarely a project history file available for reference. Furthermore, in organizations where there is a generally inadequate understanding of the planning process and its value, lack of commitment for time and resources also affect estimation results. A project management software which at least provides a foundation for such a database is needed [Zells, 1990].

As mentioned above there are two aspects to the successful completion of a software project: techniques and planning, and there are several tools to automate the techniques part, in particular, the CASE tools which support the Programmer/Analyst/Designer. However, the automation of methodologies, along with planning and management aspects, usually are not supported in a satisfactory way. This is usually accomplished manually by project managers, which makes manipulation and maintenance cumbersome. A tool to perform these management aspects could increase the productivity of the project manager, while providing a proper foundation for the project right from the start. There are stand-alone tools that allow the user to perform project estimation. Currently, support for the Project Manager is limited to such single-purpose tools. There exists a need for an integrated set of tools with the capability of providing project management support.

Estimation is a crucial aspect of project management, yet most project management software gives little recognition for this need. Each project can have several estimating models each of which must be tuned according to software project needs and must be customized. When managers try to estimate a project without being able to reference an internal corporate estimating history file, a methodology checklist, or even a book, they may end up making inaccurate estimates. Unless they have some pre-existing models or templates of project estimates, this first step may be very difficult initially. In other words, an existing estimating model helps in this first and most important step [Zells, 1990].

Objectives of the Study

The objective of this study is to design, implement, and test a Computer-Aided Project Engineering (CAPE) tool for automating and maintaining software estimation from a project management perspective. The CAPE estimating tool enables the user to load and manipulate estimating models and templates, view the estimates from various perspectives, as well as providing on-line method help (hyper-media based), and an export link to scheduler packages. The emphasis is given to the design of the user-interface component of this tool, keeping in mind the IBM SAA/CUA standard [IBM SAA/CUA standards, 1990]. The tool being

developed is called Project Engineer ESTIMATOR.

The ESTIMATOR is designed to be generic in functionality, i.e., it is independent of the estimation model. ESTIMATOR enables the user to save both models and estimates with all the relevant planning and management information for future use as estimation templates. The ESTIMATOR uses Multiple Document Interface (MDI) child windows to display information to the user from different perspectives. Concisely, Project Engineer ESTIMATOR is designed to support a wide range of estimating process management activities in a modular and integrated fashion and provide a good user interface. Function-point analysis is not supported currently, but the next enhancement will have function-point analysis capability. At present only the object based estimation method is supported.

CHAPTER II

CONCEPTS USED IN ESTIMATOR

Introduction

The purposes of this project are; a) to provide a better way to enhance the accessibility of estimating methods, b) to provide a way to tune an estimating model, c) to automate the work of the Project Manager, and d) to provide a consistent user-friendly (GUI) front end. The following sub sections explain the concepts used to realize the above mentioned purpose.

Estimating Models

Each activity in the WBS is associated with an estimated value. This value is calculated using a formula connected to each activity. These formulas determine the appropriate objects (estimating factors) for each task and the unitary effort required for each object. An estimating model is defined by a set of estimating formulas. Each component of a formula is called an estimating factor or object. Some factors are in turn derived by another formula

known as the derivation formula which, in turn consists of factors.

The user is presented with a set of questions from which to provide the number of objects that are the basis for calculating all the deliverables required for a particular project. Some of these numbers may be known. If the diagram in Figure 1. page 14. exists, then the user can count the number of objects represented by boxes and lines. The type of objects that have known values vary according to the current stage of the project. For example, if the project is in the Construction and Testing stage, the number of programs (an object) is known. However, when the project is in the Project Initiation stage, the number of programs has to be calculated.Given below is an example of an estimating model for a part of WBS for the Analysis Stage, of which only one Step is expanded.

AN - Analysis Stage (Stage) ..... Value = 10.9
AN.AN1 - System Investigation (Step) ... Value = 6.9
AN.AN1.10 - Investigate the Current Data Architecture (Task) .... Value = 5.4
AN.AN1.20 - Describe the Required Data Model (Task) ..... Value = 1.5

AN.AN2 - Non System Investigation (Step) ..Value= 4

etc...

Formula for task AN.AN1.10 is (B1+B2) * 0.9, (Value = 5.4)

where B1 is the number of separate files

(value=4)

B2 is the number of number of users

(Value = 2)

0.9 is the adjustment factor.

Formula for task AN.AN1.20 is (A1+B0) * 0.1, (Value = 1.5)

where A1 is the number of entities (boxes)

(Value = 10)

B2 is the number of relationships (lines)

(Value = 5)

0.1 is ten per effort day.

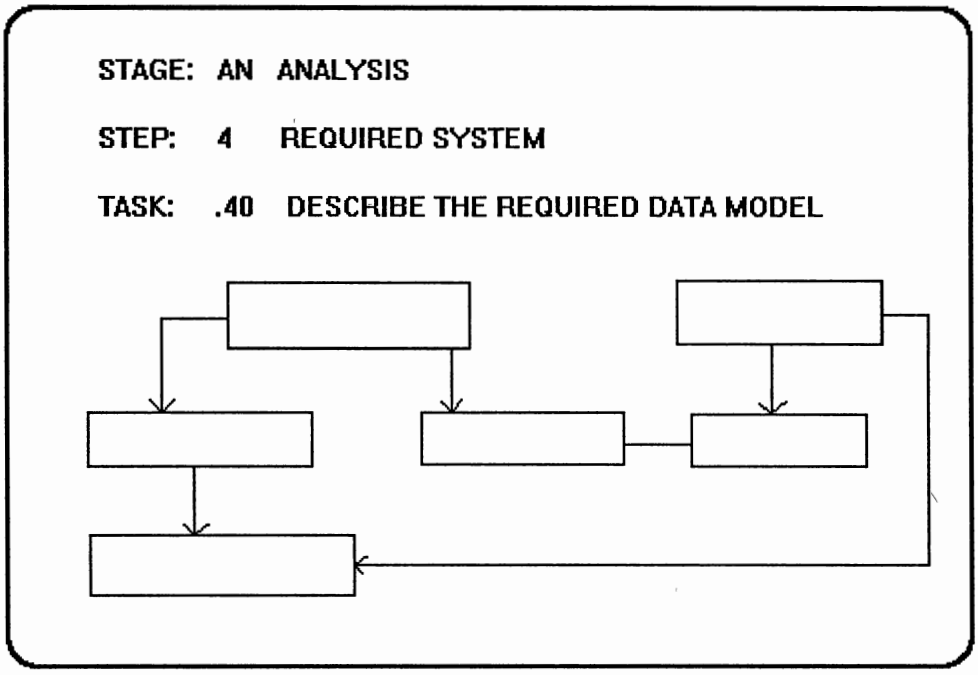


Figure 1. DFD example

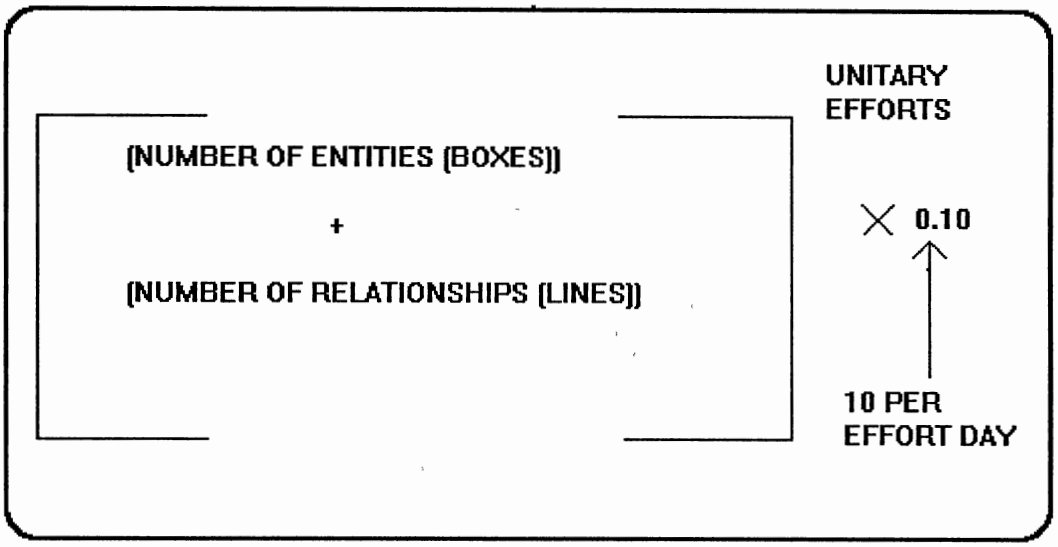


Figure 2. DFD effort estimation

Note that only the tasks have formulas associated with them, because the level of estimation is fixed at the task level. The sum of estimates at the task levels becomes the estimate for the step level. Similarly the stage estimates are computed from step level estimates. Finally, the project totals are obtained from the stages.

In later versions of ESTIMATOR, it is proposed that estimation at different levels in the WBS hierarchy be allowed. The estimation level is a property of the estimating model. The user can select only one level of estimation for a model. If varying estimation levels are allowed, the way project totals are calculated is different. In such a case, formulas are allowed only at the level at which the estimation is allowed. Level restriction means that user can manipulate formulas for only those

activities in the WBS hierarchy which are at the estimation level defined in the model. All the activities at a level lower than the estimation level obtain estimation values which are apportioned from the next higher level according to an apportionment formula. The apportionment formula is simply a percentage value. So the sum of all the apportioned values at levels lower than the estimation level is equal to one hundred percent, i.e., equal to the value for the activity at the estimation level. To calculate the estimates for all the activities which are at a level higher than the estimation level, the values at the estimation level are summed up in a bottom-up manner just like in the fixed task level estimation.

Saved Estimates

Each formula/factor can have several values. A set of formula/factor values, across an estimating model, constitutes a "saved estimate". Therefore an estimating model can have several saved estimates. The concept of a "saved estimate" allows the tool to have a "what if" capability. The user can assign different values to estimation factors within the same model, perform a calculation and see if the estimates generated are acceptable. The user can save the generated estimates for later comparisons against other estimates saved previously. The attributes associated with the object "saved estimate"

are save date, name of the estimate and the name of the person saving the estimate. The user can view these estimates from different perspectives.

There are four categories of estimating factors. They are as follows:

1) Override - Manually entered by the user.
2) Derived - Calculated from a derivation formula.
3) Other - Factor value read from other CASE tools.
4) Default - A default value is provided with a model.

The above values are used in a hierarchical order by the ESTIMATOR. The Override value has the highest precedence following the order in which the other value categories are delineated above. At least one value category must be defined and only one of the above categories is used in estimation. The user is allowed to select the category of value to be used in calculations. The derived value exists whenever a factor has a derivation value.

Global and Local Values

Each estimating factor has a value assigned to itself. A factor value can be of two types: Global and Local. Each factor has a mandatory global value. The global value of a factor is used by default in a formula calculation. But there may be situations where it is necessary to use a value other than the global value in a particular formula

calculation. In such situations, a factor is allowed to have a value that is local to that particular formula.

The concept of local and global values for a factor gives the user added functionality, and flexibility in calculating estimates.

The Project Database

CASE tools handle data that are related in complex ways. They need data integrity and non-redundancy in representation. A data repository is suited best for such an application. A repository is a mechanism for storing and organizing all information concerning a software system, including planning, analysis, design, implementation and project management information [McClure, 1989]. The purpose of a repository is to store system information at a central place, keep the data uniform, and be accessible to all users. The repository must be robust enough to cater to the needs of large software projects and must also be scalable [McClure, 1989].

One of the most essential steps in software development is data modeling. The basic concepts in data modeling are explained below.

The Metamodel

A metamodel (or metadefinition) consists of a set of OPRR constructs (entity, property, relationship, and role types), and the rules governing their use. The set of types and rules are stored in a metadatabase. A particular tool is determined by the contents of its metadatabase, which consists of [OPRR API 1991]:

-Set of entity types such as processes, stores, externals, and elements.

-Set of properties such as process number, store layout, element value.

-Set of relationship types such as data "flowing" between two processes, or the existence of a store "satisfying" a requirement.

-Roles such as "flow-part", "source-part", and "destination-part" for a "flows" relationship.

-Rules such as:"a process may only be assigned one process number", "flows may not carry information between two externals", and "entity names must be unique".

A metadatabase is created by a metalanguage description of the model. The metadatabase identifies each metaentity (i.e., object, property, relationship, or role type) by a name (metaname). The set of metanames are unique within a relationship.

Each metaname has a numerical code assigned to it.

The numerical codes are unique over the entire set of codes. The numerical code is used in the instance database.

The OPRR data engine actually uses two kinds of databases in its operation: the Metadatabase and the Instance database. An Instance database contains the actual data about a project. The OPRR database can be accessed using an interface called the Application Program Interface (API).

There are several data models employing the concepts of entities and relationships. Some of these models are delineated below [Welke, 1989].

Database Models

The following is a brief description of some of the database models currently in use [Chandrashekhar, 1991]:

The Binary Model: This is a simple model characterized by the involvement of exactly two entities and a single relationship between them. There are two types of binary models: binary-1 and binary-2. The binary-1 form allows multiple instances of only one object type, usually denoted as 1:M (one to many). Binary-2 form allows multiple instances of both entity types (M:M, many-to-many). Because of their simplicity, binary models have very limited applicability to complex applications such as CASE tools.
The Entity-Attribute-Relationship-Attribute (EARA)Model: To

overcome the limitations of the binary model and to express multi-part relationships, the EARA model is used. In this model, properties or attributes are associated with each entity participating in a relationship. Even though the EARA model allows sophisticated multi-part relationships, it does not express clearly the complexity involved in the relationship. Thus, there is a need for a higher metamodel.

The OPRR Model

To overcome the limitations of the EARA model, the OPRR model is used. In the OPRR model, roles are associated with entities that modify the way an entity takes part in a relationship. This adds another degree of freedom and hence enhances the expressiveness of the model. A metamodel is a powerful modeling technique used to capture clearly and unambiguously, capture the meaning of design notations. In other words, a metamodel is a "database schema" for a data model. Thus there is no data redundancy and the semantics are expressed clearly ensuring data integrity.

In the OPRR database, data modeling can be done at a metalevel. Hence, many-to-many binary relationships with roles attached to each participating entity and each entity having multiple properties can be expressed very easily. Two or more objects can acquire different roles and participate in one or more relationships with multiple instances. This gives additional degrees of freedom in data

representation compared to other data modeling techniques explained in the previous sections. The complete representation is can be modified easily without affecting other relationships and entities. Welke makes a comparative study of various modeling techniques and demonstrates the superiority of the OPRR technique [Welke, 1988].

Thus the metadatabase approach imparts flexibility by supporting multiple methodologies and lets the user customize his/her own methodology [Welke, 1989]. Also it fully supports future evolution of the project and lets the user add more analysis and reporting functions.

The User Interface

Recent studies have shown that the user interface forms a significant part of any application [Myers, 1988]. The user interface of any software package is that part which accepts input, interacts with the user, and presents him/her with a friendly environment. It must be designed in such a way that it makes the interaction between the user and the system easy and intelligent. There are different styles in which user and system interaction can be made; i.e., menu selection, form fill-in, command language, natural language, and direct manipulation [Shneiderman, 1987]. Of these, direct manipulation is the most popular, because direct manipulation involves selecting an object of interest and performing the required action. Elaborate graphics, ease of

use, and semantic feedback all make direct manipulation the most used interaction style for currently developed software packages [Chandrashekhar, 1991].

The design of the user-interface for this project is generally inspired by the IBM SAA/CUA standards. It is implemented to be the direct manipulation interface design along with the form fill-in approach for certain parts. The platform selected to design and implement this package is Microsoft Windows version 3.0 which features a mouse-integrated graphical user interface.

Project Engineer ESTIMATOR uses Multiple Document Interface (MDI) windows to display information. MDI child windows are windows which are controlled by and appear within a Parent or 'main' window [Petzold, 1990]. These child windows function exactly like any other window, but they are limited to the boundaries of the parent window. Each MDI Child window performs a function and can be represented by an icon.

Estimating Model Templates

Different estimating models have different sets of formulas in estimating each activity and different methods of calculating the project totals. Depending on the project characteristics such as size of the project, type of application, etc., a suitable estimating model needs to be selected. For example, a large real-time military software

project may have a large number of formulas having dependencies in a complex way, while a small commercial software project may have a much simpler set of formulas. Therefore, several "Estimating Model Templates" will be provided in Project Engineer ESTIMATOR for the user to cater to the different profiles of software projects. These templates provide a starting point for the user. Templates can then be appropriately tuned to build required estimating models.

Two types of databases exist in Project Engineer ESTIMATOR: estimating model templates and project estimating models. An estimating model template is always used as a basis for a project estimating model. It is modified only when changes that affect all future uses of the template are being made. The user is allowed to make changes to an estimating model template or the project estimating model, and to save it as a template. This facility is useful if the industry handles projects which follow a pattern but differ slightly. In order to create a new project estimation model, a template is loaded, modified, and then saved as a project estimation model.

The need for an estimation model template type of application occurs when certain groups of estimation activities must be used over and over again within the same project or across several projects [Zells, 1990]. Estimation model templates bring with them all the valuable information that is needed to give the user a head start.

Since templates are proven software estimates themselves, they bring the expert's experience and expertise with them.

CHAPTER III

DESIGN AND IMPLEMENTATION

Introduction

Project Engineer ESTIMATOR is coded in C, using the Software Development Kit supplied by Microsoft Corporation to develop applications which run in Windows version 3.0. The application requires Windows 3.0 as a base and runs on PC platforms.

Design Evolution

The aim of this study is to design, and develop a project estimation software primarily as per LBMS Inc. specifications. The following paragraphs clearly delineate LBMS specifications, and contributions of this study.

The following are the high level design specifications given by LBMS Inc.

1) The estimator data-model.

2) The estimator must support multiple approaches to estimation.

3) Editing and customization of estimating models.

4) The estimator must support macro-level estimating.

5) Use OPRR repository.

6) Create, maintain, and access metrics database.

7) Hypermedia based on-line help system.

8) The estimator must work in coordination with other tools such as life cycle builder, scheduler, etc.

The design, coding, testing, and the delivery of the complete estimator software was the primary contribution of this study. The other contributions of this study are:

1) User Interface :

-The estimator front end to look, feel, and function like a spreadsheet.

-Compliance with SAA and CUA standards.

-Recursive functioning of formula and factor editors.

-Application of Microsoft Windows multiple document interface (MDI) feature.

2) The in-memory data structure and algorithms (described in section 3.2.1).

3) Estimator computation engine consisting of a formula parser, a calculator, and data validation functions.

4) The concept of local and global values for a factor.

5) Detection of cyclic dependency between factors and formulas by application of topological sorting.

Implementation of the Concepts

The implementation of concepts from the previous chapter are briefly described in the following sub-section.

Implementation of the Database

The underlying data model for the Project Engineer ESTIMATOR is described as below. An estimating model is defined by a set of estimating formulas. Each component of a formula is called an estimating factor. Each formula/factor can have several values. As explained in previous sections, a set of formula/factor values, across an estimating model, constitute a "saved estimate". The ESTIMATOR data model is designed to give a "what if" capability to the user which, essentially means evaluating different estimates for different possible values of formula/factor. Some Factors are in turn derived by another formula known as the derivation formula which in turn consists of factors. Also, each WBS object is related to a formula. A factor may have a value local to a formula or it may have a value that is global to all related formulas. The estimator data model is shown in Figure 3. page 29.

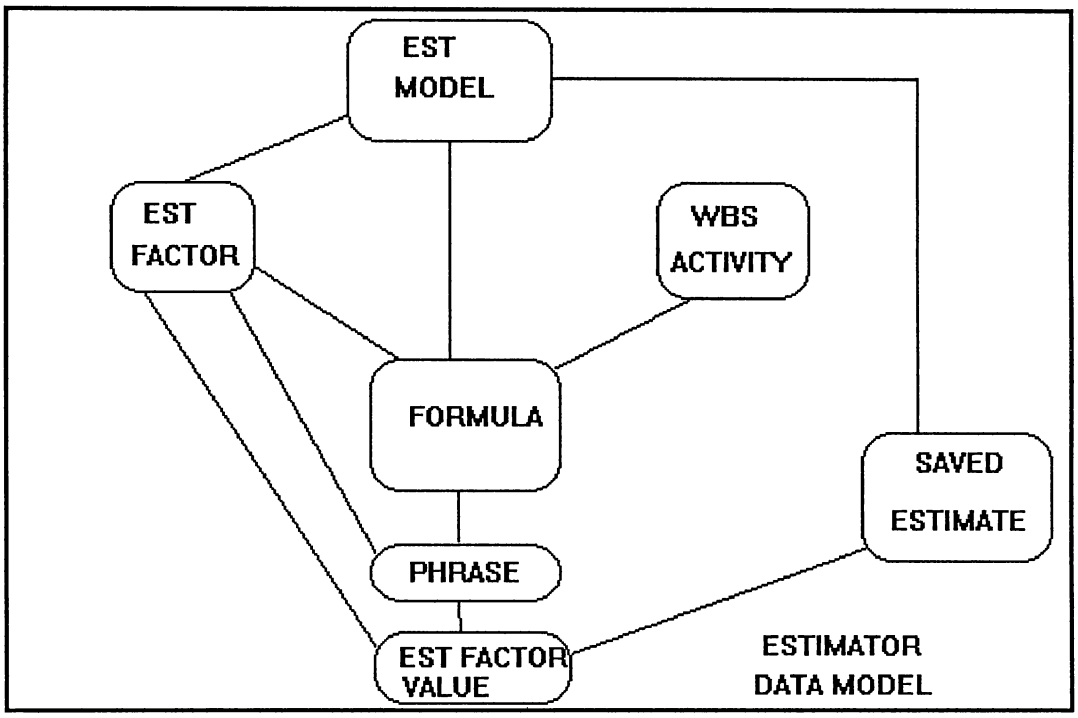


Figure 3. Estimator data model

The following objects are defined in the OPRR metamodel:

1) Estimating Model

2) Estimating Factor

3) Formula

4) Estimating Factor Value

5) Saved Estimate

The relationship between the above objects, as defined in the metamodel, are shown in Figure 4. page 30. and Figure 5. page 30.

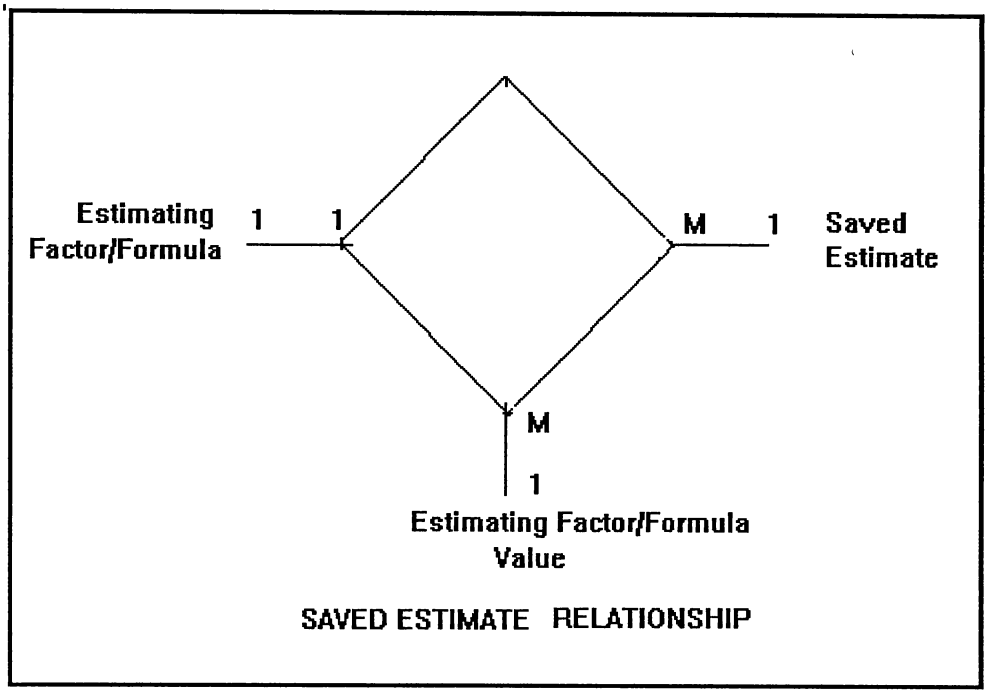


Figure 4. Meta schema of Saved Estimate

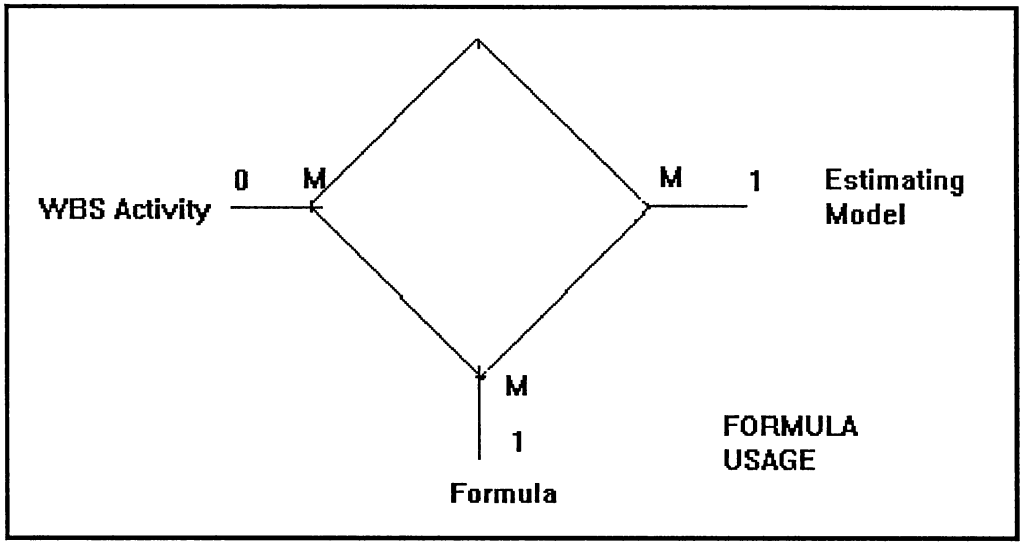


Figure 5. Meta schema of Formula object

The concepts of uniqueness and scoping are implemented for objects and relationship and are explained below [OPRR ORIF, 1990]. In a scoping relationship, two roles are important: the scoping role and the scoped role. A database object which is scoped cannot exist before the scoping relationship is created. The scoping facility allows that an identifying property need not be unique over the whole database but only within the metatype of objects to which it applies to i.e., an object within the scoped relationship in which it is defined.

The following is an example of a scoping relationship as defined in the metamodel. The relationship is called Data-Structure with two roles: a "container-part" and a "contained-part". The container-part is the scoping object. The relationship is not associated with any properties. Each role is fulfilled by an object with the specified identifying property (in this example Basic-Name is an identifying property).

```
CREATE SCOPING RELATIONSHIP Data-Structure
     ROLE
     Container-Part = ICON-1: BASIC-NAME="name of icon"
     SCOPED-ROLE
     contained-Part =
          (SELECT OBJECT Diagram: BASIC-NAME="customer-
                                   information-record");
```

The database retrieval function is one of the most

important parts of the ESTIMATOR. The retrieval builds in-memory data structure and the memory pointers between them. Figure 6. gives a psedo-code of the database retrieval algorithm.

```
                DATABASE RETRIEVAL ALGORITHM

      Retrieve an estimating model.
      Retrieve a saved estimate.

      While WBS activities
      Begin
        Retrieve WBS activity.
        Retrieve WBS-FORMULA relationship.
LABEL1:Retrieve formula object and its properties.
        Retrieve FORMULA-SAVEDEST-VALUE relationship.
        Retrieve FORMULA-FACTOR relationship.

        While there are factors for this formula
        Begin
          Retrieve factor and its properties.
          Retrieve FACTOR-SAVEDEST-GLOBALVALUE relationship
          Retrieve FACTOR-SAVEDEST-LOCALVALUE-FORMULA relationship
          Retrieve Factor value properties.
          Retrieve FACTOR-DERIVATIONFORMULA relationship.

        If Derivation formula exists for this factor then
          Goto LABEL1.
      End While.
    End While.
```

Figure 6. Database retrieval algorithm

Estimating Models Implementation

The underlying data structure is designed to be general enough to handle both different estimating models and saved estimates. The selection of models and saved estimates is accomplished by selection of a menu item, and then displaying a dialog box with model names. The name of the model and saved estimate are obtained from this dialog box. These names are sent as parameters to the data retrieval routines. The data retrieval routines retrieve only those formulas, factors, and their values that constitute the currently selected model and saved estimate. The in-memory data structure is then populated with data from the database. Only one estimating model and saved estimate is allowed to be manipulated at a time. Allowing more than one model to be used concurrently creates complex memory problems. If another model or saved estimate is chosen for manipulation, the existing model is closed by freeing all the allocated memory and reinitialising all the control variables. Then the retrieval routines are invoked with the new model, and saved estimate names as parameters.

Each WBS activity record has a pointer to one formula record. Upon changing a model, this pointer points to a new formula record. The estimated values are stored with each WBS activity record, formula record, and factor record. Therefore, each time a new saved estimate is chosen all these values are re-populated.

The User Interface Implementation

The ESTIMATOR displays data in a tabular fashion. The tabular data window has much functionality built into it and functions like a grid. The grid-like window will be called the list window henceforth. The ESTIMATOR uses the 'Grid Class' developed by LBMS Inc. to implement the list window. The list window is a scrollable child window. Several user interface designs were evaluated. Finally the decision was made to use a Grid Window Class because it is more user-friendly and has been accepted by the users (in Microsoft Excel). The Grid Window Class provides an interface for displaying and managing tabular data. Since the data associated with projects has to be displayed from various perspectives with each row of objects/tasks representing a WBS object, or a formula object, or a factor object, a grid structure is appropriate. Columns indicate the Estimation information. Columns can be customized by the user to view what the user wants at any point in time. The columns are resizable with the use of a mouse. It is essential to be able to accommodate the variety of data that can be displayed in each column. Scroll bars are provided to scroll the grid vertically or horizontally [Chandrashekhar, 1991].

Rows and columns can be 'highlighted'. Simultaneously, the contents of the selected cell are displayed in an edit window where it can be modified. Three instances of the

grid class are used to display three list windows. All three windows have different data. Therefore, the data retrieval (from database and in-memory data structure) routines are independent of the window they are servicing. All three windows have the same basic structure with minor differences in specific features. Both window-specific data and functionality are implemented by building into the code the capability to recognize in which window the user is currently in. The menus for each window are separate and are allocated dynamically depending upon the window.

One of the important user interface implementations is the formula/factor editor. The 'double-click' message is trapped and processed for popping up the formula/factor editor. These editors are modal dialog boxes created using the Microsoft Windows Dialog editor. These dialog boxes are recursive, i.e., it is possible to traverse from a formula editor to factor editor and vice versa. This means all the data to be displayed in these dialog boxes is pushed on to the stack recursively. In order to avoid the attendant memory problems, global memory is allocated dynamically for each instance of the editor invoked. This also makes the dialog box code reentrant and ensures data integrity.

The Software Architecture

The architecture employed for this project is described below. As usual the user interface forms the front-end and

uses Windows version 3.0. The metadefinition best explains the underlying software architecture. The following is the metaschema of the ESTIMATOR.

Meta Schema Implementation

WBS Activity

Properties:

COLLAPSE-EXPAND: -INTEGER- This is a flag used for display purposes, which tells us whether we should suppress all the descendants of this activity or not.

EFFORT-ESTIMATE-ORIGINAL: -INTEGER- This is the original estimate of the effort for this activity.

EFFORT-ESTIMATE-REVISED: -INTEGER- This is the revised estimate of the effort for this activity.

ACTIVITY-LEVEL: -INTEGER-

Description:This object represents each activity in the project. These objects exist in the database according to a hierarchy called the Work Breakdown Structure (WBS). This structure is built and maintained by the property of each activity called ACTIVITY-LEVEL.

Estimating Model

Properties:

DEFAULT-WORKSHEET: -INTEGER- The worksheet to displayed in the grid initially.

ORIGINAL-ESTIMATE-WORKSHEET: -INTEGER- The worksheet designated by the user to the original estimate.
REVISED-ESTIMATE-WORKSHEET: -INTEGER- The worksheet designated by the user to the revised estimate.
ESTIMATING-LEVEL: -INTEGER- The WBS activity level with estimating formulae. It is the level at which the estimating is done in the given model.
PROJECT-EST-FORMULA: -TEXT- Formula for project level estimation.
Description: This object represents an estimating model. A model represents a mathematical description of an estimating method or approach applied to a work breakdown structure. There can be multiple models in a project database. A model is expressed as a set of factors and formulae.

Saved Estimate
Properties:
SAVEDBY: -STRING- The name of the user.
MODEL-PROJECT-TOTAL: -INTEGER- The total estimate for the whole project.
PROJECT-REVISION-VALID-UNDER:--INTEGER- Version control number for worksheets.
Description: This object represents a set of values for factors and formulae across a model. This is scoped by model. This object should be renamed Worksheet.

Formula

Properties:

FORMULA-STRING: -STRING- The mathematical expression entered by the user for this formula.

Description: This is a estimating formula used in the calculation of various estimates for the activities. Formulae can be tied up to activities. They are scoped by model. A formula mixes and matches various factors.

Factor Value

Properties:

GLOBAL-FACTOR-VALUE: -STRING- The manual value entered by the user.

Description: This is the value of each factor object based on the worksheet. This is not needed if the value is made a property of the relationship between factor and worksheet.

Estimate Factor

Properties:

EST-FACTOR-CATEGORY: -INTEGER- The code which specifies the category this factor belongs to.

FACTOR-NAME: -STRING- The name of the factor.

FACTOR-DEFAULT-VALUE: -STRING- The mandatory value for the factor. This is entered by the user when the factor is created.

Description: This represents the factors used in a formula. A factor is a quantitative expression of considerations

given in the process of estimation. They have different values based on the worksheet. They are scoped by model.

Formula-Factor

Parts: FORMULA ONE-MANY

ESTIMATEFACTOR ONE-MANY

Description: This gives us the list of factors in a given formula. Both formulae and factors are scoped by model, and in this relationship, both should be scoped by the same model. There is now no way of preventing the user from going into an orf file and violating this rule, which could lead to crashes when the estimator is run. Probably, these relationships should be rethought-out. Note that the second part below is redundant and should be deleted.

Estimating Model-Saved Estimate

Parts: ESTIMATINGMODEL ONE-ONE SCOPING

SAVEDESTIMATE ONE-MANY SCOPED

Description: This relationship gives us the list of worksheets for the given model. As noted earlier, saved estimates are scoped by model.

Factor-Saved-Value

Parts: SAVEDESTIMATE ONE-MANY SCOPING

ESTIMATEFACTOR ONE-MANY

ESTFACTORVALUE ONE-MANY SCOPING

Description: The name of this relationship is misleading. It is the value that is global and not the factor itself.

We are using this relationship now to get the value of a factor for a given worksheet.

WBS Activity-Model-Formula

Parts:	ESTIMATINGMODEL	ONE-MANY	SCOPING
	ACTIVITY	ZERO-MANY	
	FORMULA	ZERO-MANY	SCOPED

Description: This relationship connects one formula of a model to exactly one activity. Therefore one activity has different formulae for different models. This is again scoping because formulae exist only in the context of a given model and the same formula may mean an entirely different thing in another model in the same project.

Model-Factor

Parts:	ESTIMATINGMODEL	ONE-MANY	SCOPING
	ESTIMATEFACTOR	ONE-MANY	SCOPED

Description: This relationship gives us the list of all the factors in the project for the given model. The relationship is scoped because factors exist only in the context of a given model and a factor with the same name might mean something totally different in another model in the same project.

The In-Memory Data Structure

The list of WBS activities is maintained in a linked list structure known as the left-linked right-sibling tree. Each WBS activity record has a pointer field pointing to its formula record. The formula records are maintained in another doubly-linked list. The formula record has pointers to each factor used in the formula. The estimating factors are maintained in another doubly-linked list.

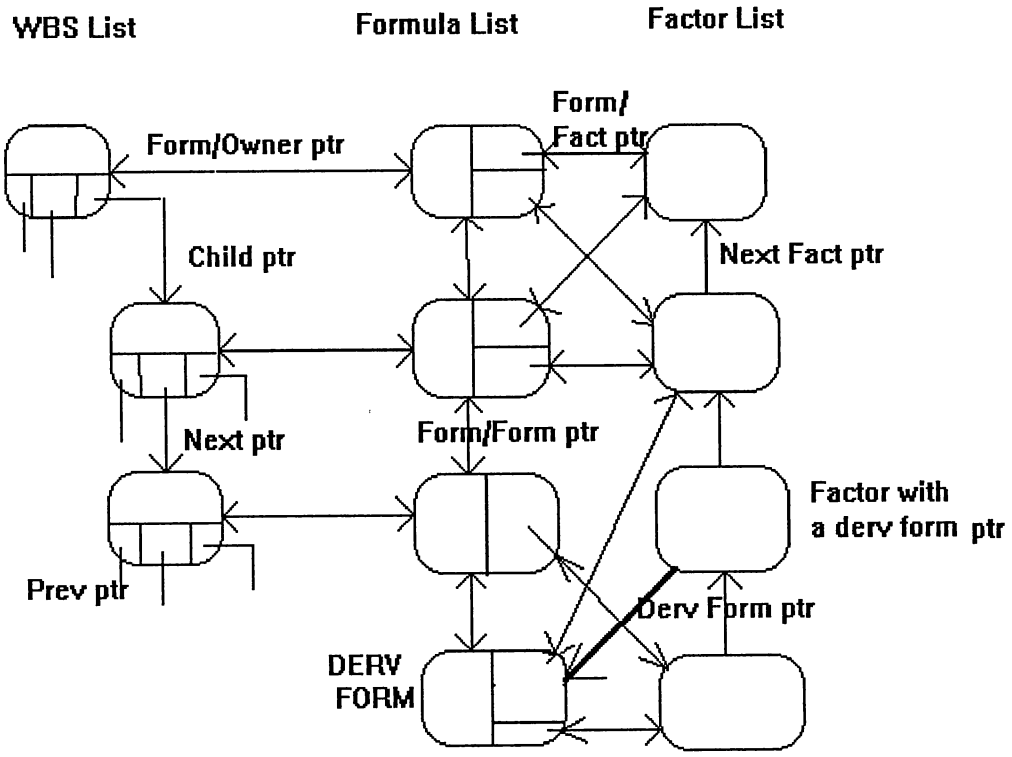


Figure 7. In-memory data structure

Each factor record maintains a linear linked list of pointers to all formulas in which it is used. This data structure scheme makes possible navigation from a factor/formula to any other related factor/formula. Extensive use of memory pointers is made to implement the complex relationships among the data elements. Figure 7. page 41. shows the relationships among various memory structures. Microsoft Windows 3.0 limits the number of memory handles to approximately two thousand. A typical project may have about two thousand WBS objects and an additional number of objects like estimating factor, formula, etc. This limits the number of separate memory handles that can be assigned to each object. To overcome this problem, an in-memory data structure is designed to improve the response time of ESTIMATOR. The in-memory data structure consists of a dynamically allocated memory block called the superblock with a memory handle. This superblock is divided logically into sub-blocks (usually 20), in which records can be stored. The address of a record is then formed by concatenating the superblock handle and the byte offset of the record. Thus, the number of handles required is reduced by a factor of 20. The Memory-Manager routine allocates and maintains a linear linked list of such superblocks for each application that uses it. For example, there is a linked list of superblocks for each record type factor, formula, and WBS activity. Upon normal termination

of the ESTIMATOR, all the superblocks are freed by traversing the linked list of superblocks.

The in-memory data structure has three primary data records viz., WBS activity record, formula record, and factor record. Each record is briefly described below.

WBS Activity Record

A WBS activity record contains all the properties of a WBS object. The properties include activity name, WBS code, WBS level, estimated value, and so on. Figure 8. page 44. depicts various fields and pointer connections of a WBS record. The following four memory pointers connect an activity to other activities and formulas.

Next Record Pointer: Points to a WBS record which is at the same level in the WBS hierarchy (sibling pointer).

Previous Record Pointer: Points back to a sibling WBS record if it exists, or to the parent WBS record (one level higher in WBS hierarchy).

Child Record Pointer: Points to a child WBS record (one level lower in WBS hierarchy) if one exists, otherwise points to null.

Formula Record Pointer: Points to a formula record if the WBS record is associated with a formula.

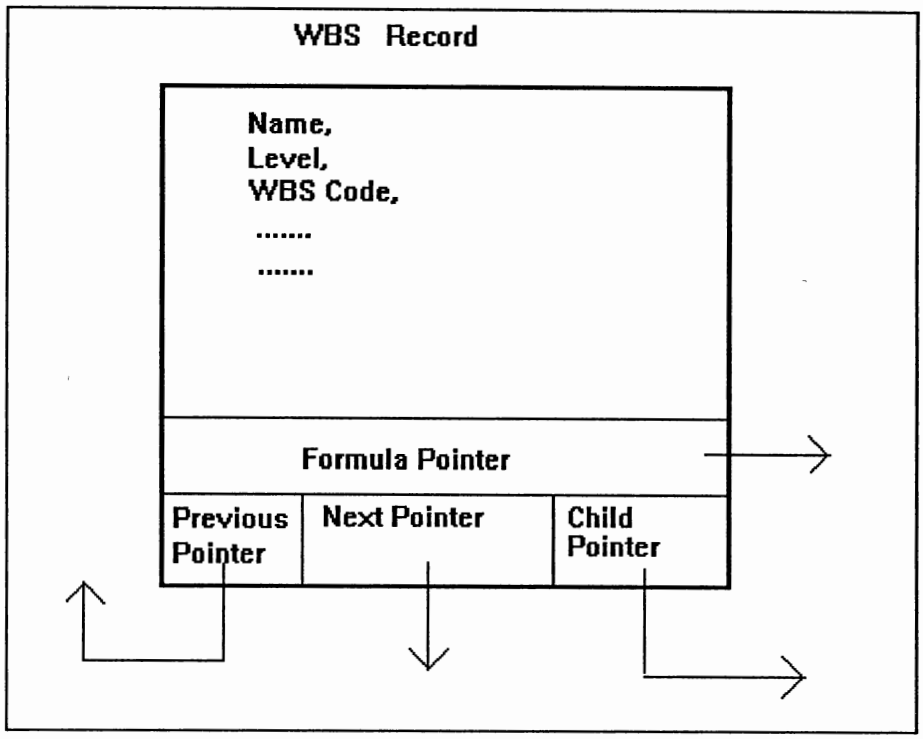


Figure 8. WBS record structure

Formula Record

A formula record contains all the properties of a formula object. The properties include formula name, estimated value, description, and so on. Figure 9. page 45. depicts fields of a formula record. The following four memory pointers connect a formula to other activities and factors.

Next Record Pointer: Points to the next formula record if it exists.

Previous Record Pointer: Points to previous formula record.

Owner Pointer: A formula is associated with either a WBS activity, or an estimating factor. The owner pointer points

to a WBS record when it is associated with a WBS activity. The owner pointer points to a factor record if the formula is a derivation formula of a factor record.

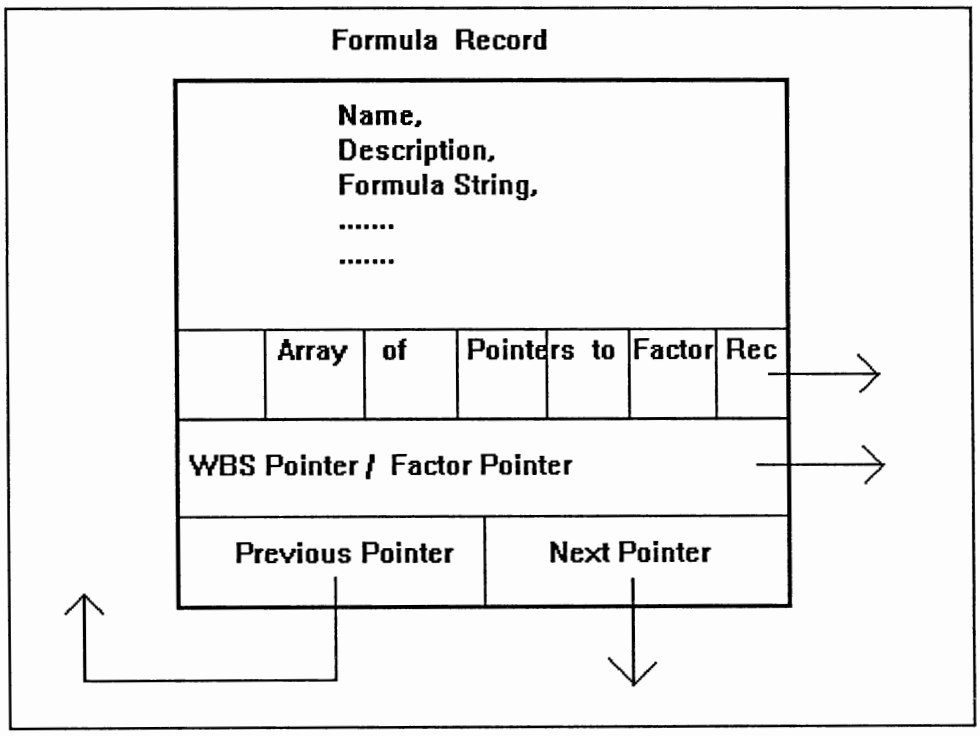


Figure 9. Formula record structure

Pointers to Factors: A formula consists of a number of factors. An array of pointers to factor records is maintained in a formula record. Each pointer points to a factor used in the formula.

Factor Record

A factor record contains all the properties of a factor object. The properties include factor name, estimated value, description, and so on. Figure 10. depicts a factor record and its ponter connections.

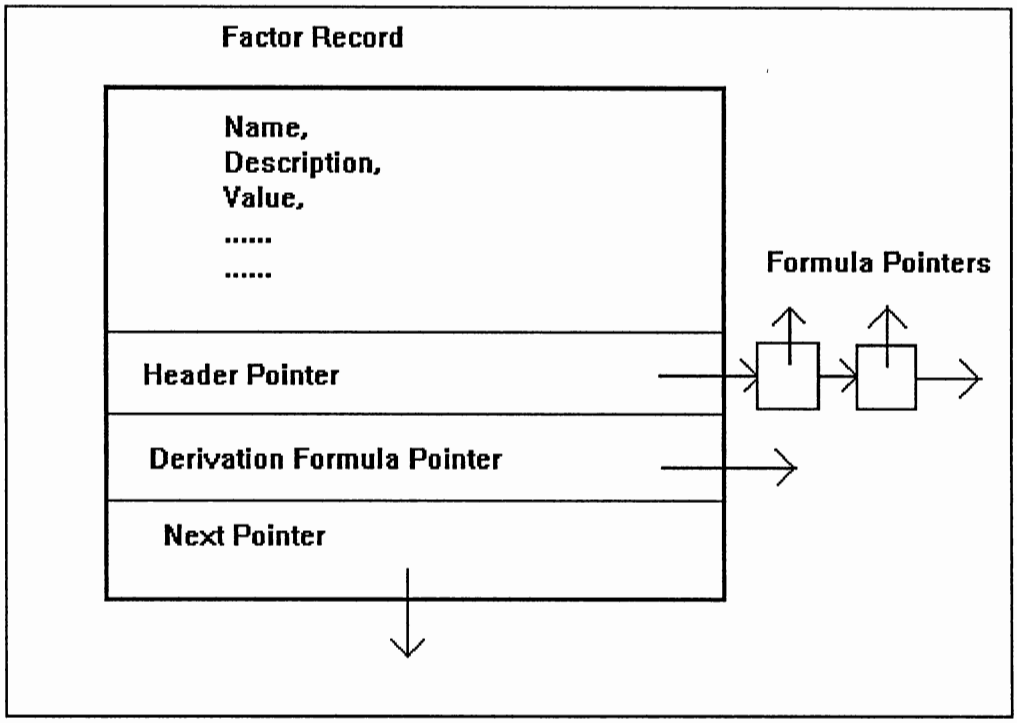


Figure 10. Factor record structure

The following three memory pointers connect a factor to other activities and formulas.

Next Record Pointer: Points to the next factor record if it exists.

Derivation Formula Pointer: If a factor has a derivation formula, then a pointer points to the derivation formula record.

Formula Linked list Header: A header node in the factor record points to a linked list of pointers to formula records. These pointers point to each formula in which this factor is used.

The ESTIMATOR Computation Engine

The concept of modular design is implemented by separating the database, the in-memory data structure, user front end, and the computation part called the Computation Engine. The Computation Engine consists of a formula parsing module, a calculator module, and a set of user service routines for navigating the in-memory data structure, creating, deleting formulas/factors, etc.

Estimation formulas entered by the user or imported from other databases, must be parsed to check for syntax. At present, only the four basic arithmetic operators are supported. It is proposed to have advanced mathematical functions such as logarithms, exponential functions, trigonometric functions, etc., in later versions of the ESTIMATOR. The parsing algorithm handles formulas with parenthesized sub-expressions and precedence constraints. A formula is first broken up into tokens (factors and operators). Each token is then validated. Parsing is

implemented by converting the formula from infix form to reverse polish form using a stack data structure and then ranking the tokens to preserve the order of precedence.

To compute the value of a formula, the formula in reverse polish notation is converted to code that is executed using a stack. Floating point arithmetic is supported. The calculator routine intelligently selects the "current" value of a factor, i.e., the formula could be using a global or local value which may again be Override, Derived, or Default value. All the factors in a formula must be defined (created) before they are used in the formula. The calculator searches the factor linked list for each factor used in a formula. If a factor is not found then it aborts the calculation giving an error message displaying the name of the undefined factor.

Circular Dependency: A WBS formula consists of estimation factors. Some factors are in turn derived by a derivation formula. The derivation formula again consists of factors. Hence formula and factors are related in a recursive manner. Such a recursion results in cyclic dependencies, which result in creating an infinite loop. To detect such cycles a topological sorting algorithm (imposing a linear order) is applied. Upon detection of a dependency the user is given an error message to correct the formula.

The Formula Editor

The purpose of this modal dialog box is to display all the information associated with one formula, and allow the user to edit any of the information fields. After a modification, the user can either accept or cancel any changes that were made. Modifications to the properties of formula require an explicit "commit" action (via an 'OK' button) before any other functions can be accessed.

The formula editor 'pops' up when the user 'double-clicks' the mouse button on any selected activity in the List window of the ESTIMATOR. The formula editor can also be accessed via the factor (if that factor has a derivation formula) editor using the "Derivation Formula" button. The formula editor validates the modified data for any error in the values input by the user. The formula string entered by the user is parsed for syntax errors. The semantic parsing of the formula is accomplished by checking to see if all the factors used in the formula are previously defined. Another function of the formula editor is to detect cyclic dependency between formulas and factors. Appropriate error message is given for the user to correct the changes made. For example, if the user enters a non-alphanumeric string for a formula field, an error Message Box is displayed requesting the user to input the data in proper format.

The formula editor is filled with the data associated with the selected formula in the List window. The user can

modify any field in this editor. The same formula editor 'pops' up when the user tries to create a 'new' derivation formula; in this case the fields are empty.

The local value of a factor can be changed in the formula editor if necessary. A new formula can be created only as a derivation formula for a factor. The formula editor for a typical formula is shown in Figure 11.

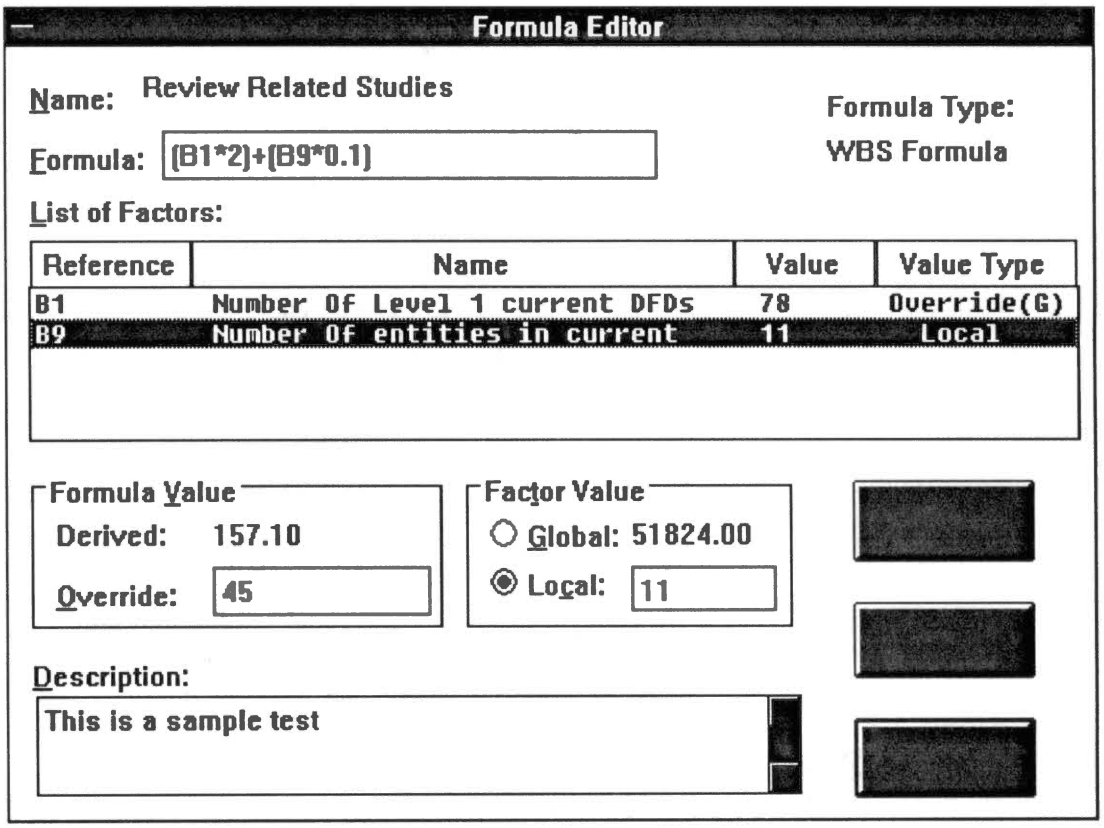


Figure 11. Formula editor

The formula editor displays a list of all factors used in a formula. The user can access the factor editor of each of these listed factors by double-clicking on the displayed factor. Similarly, the factor editor displays a list of formulas in which the factor is used. The user can access the formula editor for each of these formula by double-clicking on the displayed formula. The user can recursively traverse from formula editor to factor editor and vice versa by invoking multiple instances of these editors. Every instance of an editor (formula/factor) uses the same code. The re-entrant nature of editor code must maintain integrity of data associated with each instance of the editor. To protect the editor data, each instance maintains its own private data record. This is accomplished by sending a pointer to instance data record as a parameter to the editor function for each invocation of the editor.

Factor Editor

The purpose of this modal dialog box is to display all the information for one factor, and allow the user to edit any of the information fields. After a modification, the user can either accept or cancel any changes that were made.

The factor editor 'pops' up when the user 'double-clicks' the mouse button on any selected factor in the List window of the ESTIMATOR. This editor also validates the modified data for any error in formatting, and checks to

see if the factor is already defined. Duplicate factor definitions are not permitted.

The factor editor is filled with the data associated with the selected factor in the List window. The user can modify any field in this editor. The same factor editor 'pops' up when the user tries to insert a 'new' factor in the List window; in this case the fields are empty. Users can define a global set of categories and assign a category from this global set to each factor. Factors can then be categorized for easier maintainence of the estimating model. For example a factor can be categorized either as a productivity assumption, or a deliverable component. The factor editor for a typical factor is shown in Figure 12. page 53.

The global values of a factor can be changed in the factor editor. The local value of factor can be changed in the formula editor to which the factor value is local. The user can select any one of the four possible values for a factor viz., Manually entered, calculated (from derivation formula), imported from other databases, and default value. The user can recursively traverse from formula to factor and vice versa by invoking multiple instances of these editors. The change in the value of a factor is reflected by recalculating the values of all the formulas in which the factor is used. This recursive automatic recalculation functions similar to a spreadsheet and displays the data in a tabular form in the grid window.

Modal dialog boxes are used for formula/factor editors. The user must close all the editors (modal dialog boxes) in the order in which the editors were invoked before accessing any other item in the List window.

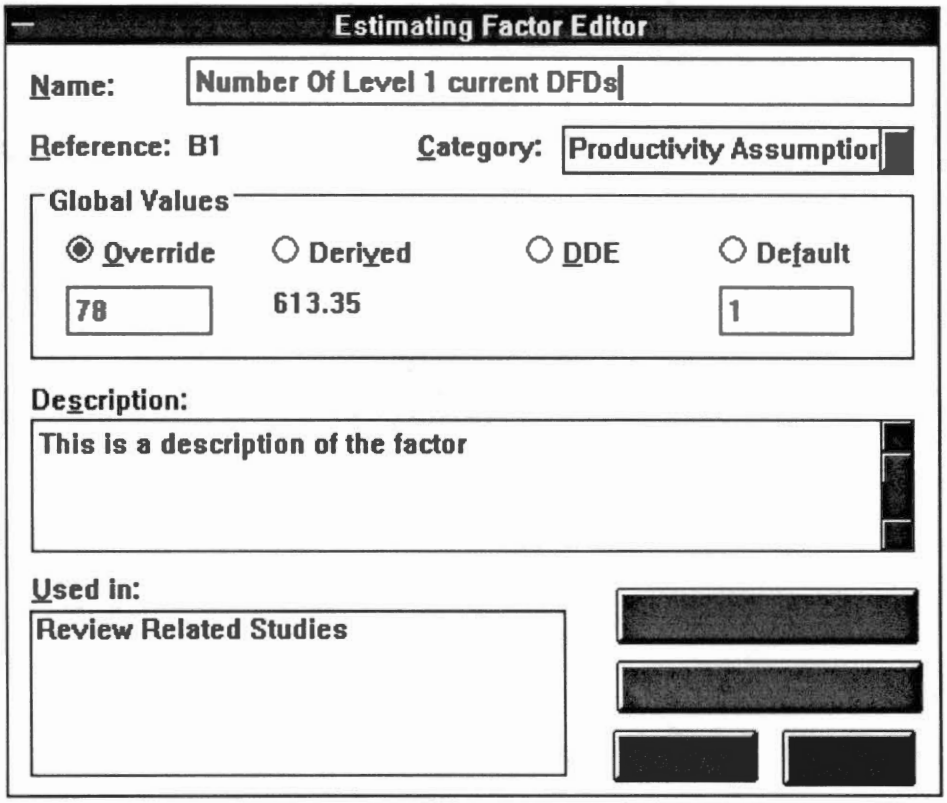


Figure 12. Factor editor

It is proposed that editors be changed to MDI child windows in future versions. The MDI child windows editors will allow mode-less access of any item in the List window.

CHAPTER IV

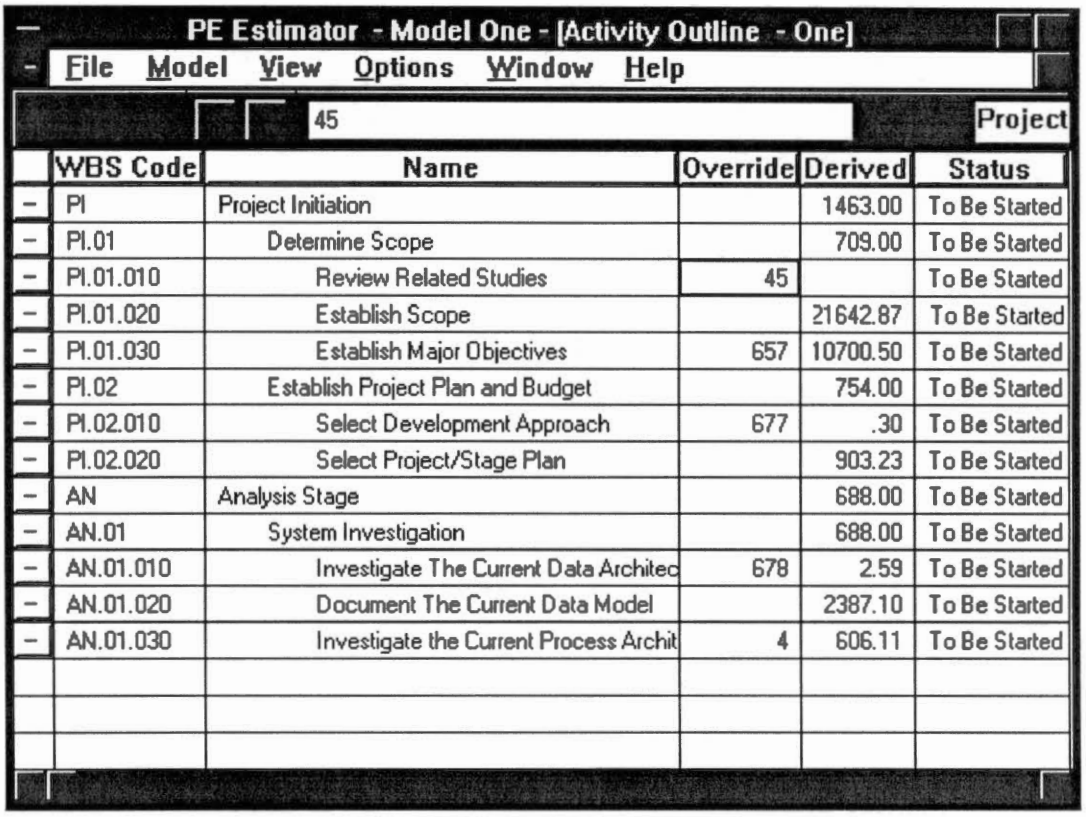
FEATURES OF THE ESTIMATOR

Introduction

The ESTIMATOR module provides estimating services for Project Engineer. It relates to all other tools like scheduling, risk analysis, etc. The purpose of the ESTIMATOR is to allow users to load and manipulate project-estimating models and model templates. The following are some of the features of the ESTIMATOR:

-Load a project-estimating model or an estimating model template.
-Load/Save saved estimates.
-Display various views of the ESTIMATOR like formula/factor, by category.
-Print the estimating information at various levels of detail
-Insert, Delete, Modify, Copy, Paste formula/factors.
-Parse/Calculate formula.
-On-line help (hyper-media based).
-Maintain information about saved estimates.
-Explode a formula/factor to a detailed description.
-Insert/Delete/Edit formula/factor categories/types.
-Allow the user to customize the appearance of the estimating model with fonts, etc.

Figure 13. is an example of a List window with WBS activities and associated information displayed. Some of the above mentioned features are shown in the menu bar of Figure 13.



	WBS Code	Name	Override	Derived	Status
-	PI	Project Initiation		1463.00	To Be Started
-	PI.01	Determine Scope		709.00	To Be Started
-	PI.01.010	Review Related Studies	45		To Be Started
-	PI.01.020	Establish Scope		21642.87	To Be Started
-	PI.01.030	Establish Major Objectives	657	10700.50	To Be Started
-	PI.02	Establish Project Plan and Budget		754.00	To Be Started
-	PI.02.010	Select Development Approach	677	.30	To Be Started
-	PI.02.020	Select Project/Stage Plan		903.23	To Be Started
-	AN	Analysis Stage		688.00	To Be Started
-	AN.01	System Investigation		688.00	To Be Started
-	AN.01.010	Investigate The Current Data Architec	678	2.59	To Be Started
-	AN.01.020	Document The Current Data Model		2387.10	To Be Started
-	AN.01.030	Investigate the Current Process Archit	4	606.11	To Be Started

Figure 13. WBS list window

The estimating factors list window is shown in Figure 14. on page 56. Observe the menu items that provide the complete set of functionality for the estimator. The Factors window provides a convenient means of editing estimates. by displaying them in a tabular fashion in the grid window.

PE Estimator - Model One - [Estimating Factors List]

File Model Edit Options Window Help

23694

	Name	Reference	Value	Category	Value Type
-	Number In Analysis Team	B13	34	Productivity Assumption	Override
-	Number Of Level 1 current DFDs	B1	1.50	Deliverable Component	Derived
-	Number of expected seperate su	A19	1	Deliverable Component	Default
-	New system multiplier	B0	23694	Productivity Assumption	Override
-	Number Of entities in current	B9	51824.00	Deliverable Component	Derived
-	Number of computer master and	A10	1	Productivity Assumption	Default
-	Number of different types of m	A13	2034	Productivity Assumption	Override
-	Adjustment Factor	C1	2	Productivity Assumption	Override
-	Number of seperate user areas	B8	26734	Deliverable Component	Override
-	Additional Screens required	A21	49	Productivity Assumption	Override
-	Number of system functions	B16	34.50	Deliverable Component	Derived
-	DFD Adjustment Factor	B2	1	Productivity Assumption	Default

Figure 14. Factors list window

How to Use ESTIMATOR

The ESTIMATOR is designed to be both consistent and easy to use. The ESTIMATOR is started by 'double clicking' on the ESTIMATOR icon. From the initial menu, the user can select the project to be estimated from a Project-Open dialog box. Then the ESTIMATOR reads data from the database for the selected project. Initially two LIST windows are displayed, one each for WBS activities and estimating factors. Because these two windows are the most frequently used by the user, they are displayed initially, by default.

A third List window for formulas can be viewed by selecting the Formula-View menu item. One list window displays WBS objects and their related formulas. Another list window displays only estimating formulas. The third list window displays estimating factors by category (productivity assumption, adjustment factor, etc.). Each of these List windows display the estimating values for each item displayed along with the value category.

Entries (formula/factors) in the window can be assigned different fonts and sizes based on styles. The list window also provides column configurability. The list window supports character attributes determined by the type of information to be displayed.

To edit/create a formula/factor, a MDI child window known as the formula/factor editor is provided. It displays all the information for one formula/factor, and allows the user to edit any of the information fields. The formula/factor editor pops up when the user double-clicks the mouse button on any selected formula/factor in the list window. It also validates and parses the data input by the user. The same formula/factor editor pops up when the user tries to insert a new formula/factor in the List window.

Upon changing the value of an estimation factor, the recalculation of all the dependendent formulas (both for activities and derivation factors) is done automatically. The updated values are displayed in the List window. To calculate the estimates at the project level the

'Recalculate Now' item must to be selected. The project totals can be viewed by selecting 'Model' item from the menu.

The ESTIMATOR is designed to be used by the project managers to create and maintain estimating models. Estimating models can be created from existing templates or afresh by selecting an empty model template. To start with, the user can open a new project life-cycle by selecting an appropriate model that might closely match the new project. Then appropriate formulas are assigned to tasks along with their tentative estimated values. The edited model can be saved, either as a model or a model template.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

Conclusions

Most project management software packages assume that estimation is done off-line. The ESTIMATOR fulfills the need for on-line estimation, and also enables customization of estimating models, supports project management activities, and provides a consistent user-friendly environment. Project Engineer ESTIMATOR fulfills the above mentioned needs that a Project Manager requires. The ESTIMATOR tool provides a generic model-independent tool to manage the process of estimation with a user-friendly front end.

This work does not go into merits or demerits of different estimating models or methods. Nor does it propose any new ways of estimation. At present the ESTIMATOR is partially functional, but it demonstrates the concept of CAPE well.

Future Work

The future versions of the ESTIMATOR is proposed to have the following enhancements:

-Estimating models can perform estimation at different levels of the WBS hierarchy. The user selects the level at which estimation is to be performed for a particular model. This enhancement will require changes/additions to the algorithms that calculate the project estimates.

-Advanced mathematical functions like trigonometric functions, exponential functions are proposed to be added to the present repertoire of estimation functions. This will require substantial additions the Computation Engine module.

-The user must be able view saved estimates (estimated values) for different models simultaneously, and be able to perform statistical comparisons. It is suggested that the estimator have such a capability in which the user can retrieve several saved estimates across different models and perform statistical functions like calculating mean, standard deviation, and so on.

-Implement user configurable estimating units. Provide conversion of different estimating units.

SELECTED BIBLIOGRAPHY

AD/M Productivity Measurement and Estimate Validation, IBM CI/S & A Guideline 313, January 1985.

Burrill C., and Ellsworth L., Modern Project Management, Burrill-Ellsworth Associates, Inc., New Jersy, 1980.

Function Point Analysis With LSDM, LBMS Plc., London, 1988.

Hsieh, D., Personal communications with David Hsieh, 1990.

IBM Systems Application Architecture - Common User Access Advanced Interface Design Guide, International Business Machines Corp., 1989.

LBMS Systems Engineering Estimating Guidlines, LBMS Inc., Houston, 1989.

LBMS Systems Engineering Methods Handbook, LBMS Inc., Houston, 1990.

McClure, C., CASE is Software Automation, Prentice Hall, New Jersey, 1989.

Microsoft Windows Programmers Reference for Windows 3.0, Vol. 1 and Vol. 2, Microsoft Corporation, 1990.

Mohanty, Siba., "Software Cost Estimation: Present and Future", Software Practice and Experience, pp. 103-120, November 1981.

Myers, B. J., "Creating User Interfaces by Demonstration", Academic Press, Inc., California, 1988.

OPRR Interchange Format (ORIF) Language Specification, Meta/LBMS Inc., Ann Arbor, 1990.

Petzold, C., "A New Multiple Document Interface API Simplifies MDI Application Development", Microsoft Systems Journal, pp. 53-63, July 1990.

Pressman, R. S., Making Software Engineering Happen, Prentice Hall, New Jersey, 1988.

Reifer, Donald. J., "CASE and Software Cost Estimating", CASE Trends , pp. 11-16, January/February 1991.

Shneiderman, B., Designing the User Interface Strategies for Effective Human-Computer Interaction, Addison-Wesley Publishing Company, 1987.

Specification for OPRR Application Programming Interface, Version 2, Meta/LBMS Inc., Ann Arbor, 1991.

Chandrashekhar, S., Personal communications with Chandrashekhar Sridhar, 1990-91.

Chandrashekhar, S., An Integrated Set of Tools to Assist in the Development and Maintenance of Project Life Cycles, Oklahoma State University Masters degree thesis, 1991.

Welke, R. J., "Meta Systems on Meta Models", CASE Outlook, pp. 35-43, Vol. 4, 1989.

Zells, L., Managing Software Projects, QED Information Sciences, Inc., Massachusetts, 1990.

VITA

Gopal N. Kulkarni

Candidate for the Degree of

Master of Science

Thesis: A TOOL TO AUTOMATE SOFTWARE PROJECT ESTIMATION FROM A PROJECT MANAGEMENT PERSPECTIVE

Major Field: Computer Science

Biographical:

Personal Data: Born in Bijapur, India, July 22, 1960, the son of Narayanrao D. Kulkarni and Laxmi N. Kulkarni.

Education: Received Bachelor of Engineering Degree in Mechanical Engineering from Karnatak University, India in February 1984; completed requirements for the Master of Science degree at Oklahoma State University in December, 1991.

Professional Experience:Research Assistant, Department of Computer Science Oklahoma State University, August,1990, to May,1991.
Part-time Programmer,Department of Agricultue, Oklahoma State University, March, 1989, to July,1990.
Assistant Engineer, WIDIA (India) Limited, Bangalore, India, June, 1985, to June 1988.
Trainee Engineer, Mysore Kirloskar Limited, Hubli, India, May, 1984, to April 1985.